Face Recognition Vendor Test Ongoing

General Evaluation Specifications

VERSION 2.0 DRAFT

Patrick Grother
Mei Ngan
Kayee Hanaoka
Information Access Division
Information Technology Laboratory

Contact via frvt@nist.gov

February 18, 2022

National Institute of Standards and Technology
U.S. Department of Commerce

Revision History

Date	Version	Description
April 1, 2019	1.0	Initial document
September 9, 2020	1.1	Update operating system to CentOS 8.2 and compiler to g++ 8.3.1 Adjust the legal similarity score range
February 14, 2022	1.2	Update operating system to Ubuntu 20.04.3 and compiler to g++ 9.3.0
February 18, 2022	2.0	Addition of Media structure (section 8.2) to support still and video imagery

Table of Contents

1. Audience 3
2. Rules for Participation 3
 2.1. Participation Agreement 3
 2.2. Validation 3
 2.3. Number and Schedule of Submissions 3
3. Reporting 3
 3.1. Version Control 3
4. Hardware specification 4
5. Operating system, compilation, and linking environment 4
6. Software and Documentation 5
 6.1. Library and Platform Requirements 5
 6.2. Configuration and developer-defined data 5
 6.3. A Note on Training 5
 6.4. Submission folder hierarchy 5
 6.5. Installation and Usage 6
 6.6. Documentation 6
 6.7. Modes of operation 6
7. Runtime behavior 6
 7.1. Interactive behavior, stdout, logging 6
 7.2. Exception Handling 6
 7.3. External communication 6
 7.4. Stateless behavior 6
 7.5. Single-thread Requirement/Parallelization 6
8. Data structures supporting the API 7
 8.1. Face Images 7
 8.2. Media (still or video) 7
 8.3. Data structure for eye coordinates 8
 8.4. Template Role 8
 8.5. Data type for similarity scores 8
 8.6. Data structure for return value of API function calls 9

List of Tables

Table 1 – Structure for a single image 7
Table 2 – Labels describing categories of Images 7
Table 3 – Structure for a set of images from a single person 7
Table 1 – Structure for a piece of media 7
Table 2 – Labels describing categories of Media 8
Table 4 – Structure for a pair of eye coordinates 8
Table 5 – Labels describing template role 8
Table 6 – Enumeration of return codes 9
Table 7 – ReturnStatus structure 9

1. Audience

Participation in FRVT is open to any organization worldwide. There is no charge for participation. The target audience is researchers and developers of FR algorithms. While NIST intends to evaluate stable technologies that could be readily made operational, the test is also open to experimental, prototype and other technologies. All algorithms **must** be submitted as implementations of the API defined in the specific test's API document.

2. Rules for Participation

2.1. Participation Agreement

A participant must properly follow, complete, and submit the FRVT Participation Agreement. This must be done once, either prior or in conjunction with the very first algorithm submission. It is not necessary to do this for each submitted implementation thereafter UNLESS there are major organizational changes to the submitting entity.

NOTE If an organization updates their cryptographic signing key, they must send a new completed participation agreement submission for this evaluation, with the fingerprint of their public key.

2.2. Validation

Prior to submission, all participants must run their software through the provided corresponding validation package for the test they wish to enter. The validation package will be made available at https://github.com/usnistgov/frvt. The purpose of validation is to ensure consistent algorithm output between the participant's execution and NIST's execution.

2.3. Number and Schedule of Submissions

Participants may send one submission as often as every four calendar months from the last submission for evaluation. NIST will evaluate implementations on a first-come-first-served basis, and quickly publish results.

3. Reporting

Unless otherwise specified for a specific test, for all algorithms that complete the evaluations, NIST will post performance results on the NIST FRVT website. NIST will maintain an email list to inform interested parties of updates to the website. Artifacts will include a leaderboard highlighting the top performing submissions in various areas (e.g., accuracy, speed etc.) and individual implementation-specific report cards. NIST will maintain reporting on the two most recent algorithm submissions from any organization. In the event an algorithm is no longer operable (e.g., license expiration, etc.), that algorithm will be retired from the evaluation. Prior submission results will be archived but remain accessible via a public link.

Important: This is an open test in which NIST will identify the algorithm and the developing organization. Algorithm results will be attributed to the developer. Results will be machine generated (i.e. scripted) and will include timing, accuracy and other performance results. These will be posted alongside results from other implementations. Results will be expanded and modified as additional implementations are tested, and as analyses are implemented. Results may be regenerated on-the-fly, usually whenever additional implementations complete testing, or when new analysis is added.

NIST may additionally report results in workshops, conferences, conference papers and presentations, journal articles and technical reports.

3.1. Version Control

Developers must submit a version.txt file in the doc/ folder that accompanies their algorithm – see Section 6.4. The string in this file should allow the developer to associate results that appear in NIST reports with the submitted algorithm. This is intended to allow end-users to obtain productized versions of the prototypes submitted to NIST. NIST will publish the contents of version.txt. NIST has previously published MD5 hashes of the core libraries for this purpose.

4. Hardware specification

NIST intends to support high performance by specifying the runtime hardware beforehand. There are several types of computer blades that may be used in the testing. Each machine has at least 128 GB of memory. We anticipate that 16 processes can be run without time slicing, though NIST will handle all multiprocessing work via `fork()`[1]. Participant-initiated multiprocessing is not permitted.

All implementations shall use 64 bit addressing.

NIST intends to support highly optimized algorithms by specifying the runtime hardware. There are several types of computers that may be used in the testing. The following list gives some details about possible compute architectures:

- Dual Intel® Xeon® E5-2630 v4 CPU @ 2.20GHz (10 cores each)[2]
- Dual Intel® Xeon® E5-2680 v4 CPU @ 2.4GHz (14 cores each)[2]
- Dual Intel® Xeon® Gold 6140 CPU @ 2.30GHz (18 cores each)[3]

All timing tests will be measured on Xeon R CPU E5-2630 v4 @ 2.20GHz processors. FRVT tests will not support the use of Graphics Processing Units (GPUs).

5. Operating system, compilation, and linking environment

The operating system that the submitted implementations shall run on will be released as a downloadable file accessible from https://nigos.nist.gov/evaluations/ubuntu-20.04.3-live-server-amd64.iso which is the 64-bit version of Ubuntu 20.04.3 LTS (Focal Fossa) running Linux kernel 5.4.0-91-generic.

For this test, Windows machines will not be used. Windows-compiled libraries are not permitted. All software must run under Ubuntu 20.04.3.

NIST will link the provided library file(s) to our C++ language test drivers. Participants are required to provide their library in a format that is dynamically-linkable using the C++17 compiler g++ version 9.3.0.

A typical link line might be

```
g++ -I. -Wall -m64 -o frvt11 frvt11.cpp  -L.  -lfrvt_11_acme_007
```

The Standard C++ library should be used for development. Header files containing API function prototypes will be provided separately for each FRVT track and documented in each of the corresponding API documents.

The header files will be made available to implementers at https://github.com/usnistgov/frvt. All algorithm submissions will be compiled against the officially published header files – developers should not alter the header files when compiling and building their libraries.

All compilation and testing will be performed on x86_64 platforms. Thus, participants are strongly advised to verify library-level compatibility with g++ (on an equivalent platform) prior to submitting their software to NIST to avoid linkage problems later on (e.g. symbol name and calling convention mismatches, incorrect binary file formats, etc.).

[1] http://man7.org/linux/man-pages/man2/fork.2.html

[2] cat /proc/cpuinfo returns fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 fma cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch ida arat epb pln pts dtherm tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm cqm rdseed adx smap xsaveopt cqm_llc cqm_occup_llc

[3] cat /proc/cpuinfo returns fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 fma cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch ida arat epb pln pts dtherm tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm cqm mpx avx512f rdseed adx smap clflushopt avx512cd xsaveopt xsavec xgetbv1 xsaves cqm_llc cqm_occup_llc

6. Software and Documentation

6.1. Library and Platform Requirements

Participants shall provide NIST with binary code only (i.e. no source code). The implementation should be submitted in the form of a dynamically-linked library file.

The core library shall be named according to "Implementation Library Filename" documented in each API document. The library name will generally follow the convention: *libfrvt_<track>_<provider>_<sequence>.so*. Additional supplemental libraries may be submitted that support this "core" library file (i.e. the "core" library file may have dependencies implemented in these other libraries). Supplemental libraries may have any name, but the "core" library must be dependent on supplemental libraries in order to be linked correctly. The **only** library that will be explicitly linked to the test driver is the "core" library.

Intel Integrated Performance Primitives (IPP) ® libraries are permitted if they are delivered as a part of the developer-supplied library package. It is the provider's responsibility to establish proper licensing of all libraries. The use of IPP libraries shall not prevent running on CPUs that do not support IPP. Please take note that some IPP functions are multithreaded and threaded implementations are prohibited.

Developers may obviously use common deep learning frameworks (e.g. Caffe, TensorFlow, etc.) and should submit those dependencies as supplemental libraries. NIST has successfully received and run implementations leveraging such deep learning frameworks in other evaluations with no issues.

Do not include any standard libraries (e.g., libc.so, libgcc.so, etc.) that come with the operating system and/or compilation environment in your submission. The NIST test harness will handle all image I/O, so do not include JPEG or PNG libraries (i.e., libjpg.so, libpng.so) in your submission. If you need to include those libraries for other reasons, please contact NIST prior to your submission. NIST will report the size of the supplied libraries.

Important: Public results will be attributed with the provider name and the 3-digit sequence number in the submitted library name.

6.2. Configuration and developer-defined data

The implementation under test may be supplied with configuration files and supporting data files. NIST will report the size of the supplied configuration files.

6.3. A Note on Training

NIST and the FRVT program do not train face recognition algorithms. We do not provide training data to software, and software is prohibited from adapting to any data we pass to the algorithms. Training of face recognition algorithms is not a turn-key operation; instead it is typically an extended process involving researchers curating suitable training sets, establishing architectures and hyperparameters, and running trials over days or weeks, and then evaluating the output. The result of such a process, which is often iterative, is usually a "trained model" i.e. static data and parameters that can be saved and provided to NIST as an integral part of the black-box recognition engine. NIST does not support training, because our tests seek to mimic operational reality and, there, algorithms are almost always shipped and used "as is" without any training or adaptation to customer data. The representation of the face, as described by the "model", is fixed until the software is upgraded.

6.4. Submission folder hierarchy

Participant submissions shall contain the following folders at the top level

- lib/ - contains all participant-supplied software libraries
- config/ - contains all configuration and developer-defined data, e.g., trained models
- doc/ - contains version.txt, which documents versioning information for the submitted software and any other participant-provided documentation regarding the submission
- validation/ - contains validation output

6.5. Installation and Usage

The implementation shall be installable using simple file copy methods. It shall not require the use of a separate installation program and shall be executable on any number of machines without requiring additional machine-specific license control procedures or activation. The implementation shall not use nor enforce any usage controls or limits based on licenses, number of executions, presence of temporary files, etc. The implementation shall remain operable for at least six months from the submission date.

6.6. Documentation

Participants shall provide documentation of additional functionality or behavior beyond that specified here.

6.7. Modes of operation

Implementations shall not require NIST to switch “modes” of operation or algorithm parameters. For example, the use of two different feature extractors must either operate automatically or be split across two separate library submissions.

7. Runtime behavior

7.1. Interactive behavior, stdout, logging

The implementation will be tested in non-interactive “batch” mode (i.e. without terminal support). Thus, the submitted library shall:

– Not use any interactive functions such as graphical user interface (GUI) calls, or any other calls which require terminal interaction e.g. reads from “standard input”.

– Run quietly, i.e. it should not write messages to "standard error" and shall not write to “standard output”.

– Only if requested by NIST for debugging, include a logging facility in which debugging messages are written to a log file whose name includes the provider and library identifiers and the process PID.

7.2. Exception Handling

The application should include error/exception handling so that in the case of a fatal error, the return code is still provided to the calling application.

7.3. External communication

Processes running on NIST hosts shall not side-effect the runtime environment in any manner, except for memory allocation and release. Implementations shall not write any data to external resource (e.g. server, file, connection, or other process), nor read from such, nor otherwise manipulate it. If detected, NIST will take appropriate steps, including but not limited to, cessation of evaluation of all implementations from the supplier, notification to the provider, and documentation of the activity in published reports.

7.4. Stateless behavior

All components in this test shall be stateless, except as noted. This applies to face detection, feature extraction and matching. Thus, all functions should give identical output, for a given input, independent of the runtime history. NIST will institute appropriate tests to detect stateful behavior. If detected, NIST will take appropriate steps, including but not limited to, cessation of evaluation of all implementations from the supplier, notification to the provider, and documentation of the activity in published reports.

7.5. Single-thread Requirement/Parallelization

Implementations must run in single-threaded mode, because NIST will parallelize the test by dividing the workload across many cores and many machines. Implementations must ensure that there are no issues with their software being parallelized via the `fork()` function. Developers should take caution with checking threading when using third-party frameworks (e.g., TensorFlow, MXNet, etc.).

8. Data structures supporting the API

The following section documents the common data structures used to support the C++ API functions for the various FRVT tasks. The actual C++ API function prototypes themselves are documented separately for each test and are available on the website for each track.

8.1. Face Images

An individual can be represented by K ≥ 1 two-dimensional facial images.

Table 1 – Structure for a single image

C++ code fragment	Remarks
`typedef struct Image`	
`{`	
`    uint16_t image_width;`	Number of pixels horizontally
`    uint16_t image_height;`	Number of pixels vertically
`    uint16_t image_depth;`	Number of bits per pixel. Legal values are 8 and 24.
`    std::shared_ptr<uint8_t> data;`	Managed pointer to raster scanned data. Either RGB color or intensity. If image_depth == 24 this points to 3WH bytes RGBRGBRGB... If image_depth == 8 this points to WH bytes `IIIIIII`
`    Label description;`	Single description of the image. The allowed values for this field are specified in the enumeration in Table 2.
`} Image;`	

An **Image** will be accompanied by one of the labels from the **Label** enumeration declared within the **Image** structure given below. Face recognition implementations should tolerate **Images** of any category.

Table 2 – Labels describing categories of Images

Label as C++ enumeration	Meaning
`enum class Label {`	
`    Unknown=0,`	Either the label is unknown or unassigned.
`    Iso,`	Frontal, intended to be in conformity to ISO/IEC 19794-5:2005.
`    Mugshot,`	From law enforcement booking processes. Nominally frontal.
`    Photojournalism,`	The image might appear in a news source or magazine. The images are typically taken by professional photographer and are well exposed and focused but exhibit pose and illumination variations.
`    Exploitation,`	The image is taken from a child exploitation database. This imagery has highly unconstrained pose and illumination, expression and resolution.
`    Wild`	Unconstrained image, taken by an amateur photographer, exhibiting wide variations in pose, illumination, and resolution.
`};`	

Table 3 – Structure for a set of images from a single person

C++ code fragment	Remarks
`using Multiface = std::vector<Image>;`	Vector of Image objects

8.2. Media (still or video)

A piece of media can contain a vector of K >= 1 still photographs or sequential video frames.

Table 4 – Structure for a piece of media

C++ code fragment	Remarks
`typedef struct Media`	
`{`	

`    Label type;`	Type of media
`    std::vector<Image> data;`	Vector of still(s) or sequential video frames
`    uint8_t fps;`	For video data, the frame rate in frames per second
`} Media;`	

A piece of **Media** will be accompanied by one of the labels from the **Label** enumeration declared within the **Media** structure given below.

Table 5 – Labels describing categories of Media

Label as C++ enumeration	Meaning
`enum class Label {`	
`    Image=0,`	Still photo(s) of an individual
`    Video`	Sequential video frames of an individual
`};`	

8.3. Data structure for eye coordinates

The eye coordinates shall follow the placement semantics of the ISO/IEC 19794-5:2005 standard - the geometric midpoints of the endocanthion and exocanthion (see clause 5.6.4 of the ISO standard).

Sense: The label "left" refers to subject's left eye (and similarly for the right eye), such that xright < xleft.

Table 6 – Structure for a pair of eye coordinates

C++ code fragment	Remarks
`typedef struct EyePair`	
`{`	
`   bool isLeftAssigned;`	If the subject's left eye coordinates have been computed and assigned successfully, this value should be set to true, otherwise false.
`   bool isRightAssigned;`	If the subject's right eye coordinates have been computed and assigned successfully, this value should be set to true, otherwise false.

`   uint16_t xleft;`
`   uint16_t yleft;`
	X and Y coordinate of the center of the subject's left eye. If the eye coordinate is out of range (e.g. x < 0 or x >= width), `isLeftAssigned` should be set to false, and the eye coordinates will be ignored.

`   uint16_t xright;`
`   uint16_t yright;`
	X and Y coordinate of the center of the subject's right eye. If the eye coordinate is out of range (e.g. x < 0 or x >= width), `isRightAssigned` should be set to false, and the eye coordinates will be ignored.

`} EyePair;`	

8.4. Template Role

Labels describing the type/role of the template to be generated will be provided as input to template generation.

Table 7 – Labels describing template role

Label as C++ enumeration	Meaning
`enum class TemplateRole {`	
`    Enrollment_11,`	Enrollment template for 1:1 matching
`    Verification_11,`	Verification template for 1:1 matching
`    Enrollment_1N,`	Enrollment template for 1:N identification
`    Search_1N`	Search template for 1:N identification
`};`	

8.5. Data type for similarity scores

Identification and verification functions shall return a measure of the similarity between the face data contained in the two templates. The datatype shall be an eight byte double precision real.

The similarity score values should be reported on the range that is used in the developer's software products. Larger values indicate more likelihood that the two samples are from the same person. However, we require scores to be non-negative. Developers often use [0,1], for example. Our test reports include various plots with threshold values e.g. FMR(T), to allow end-users to set thresholds in operations. These plots may become difficult to interpret if scores span many orders of magnitude.

Providers are cautioned that algorithms that natively produce few unique values will be disadvantaged by the inability to set a threshold precisely, as might be required to attain a false match rate of exactly 0.0001, for example.

8.6. Data structure for return value of API function calls

Table 8 – Enumeration of return codes

Return code as C++ enumeration	Meaning
`enum class ReturnCode {`	
`Success=0,`	Success
`UnknownError,`	Catch-all error
`ConfigError,`	Error reading configuration files
`RefuseInput,`	Elective refusal to process the input, e.g. because cannot handle greyscale
`ExtractError,`	Involuntary failure to process the image, e.g. after catching exception
`ParseError,`	Cannot parse the input data
`TemplateCreationError,`	Elective refusal to produce a template (e.g. insufficient pixels between the eyes)
`VerifTemplateError,`	For matching, either or both of the input templates were result of failed feature extraction
`FaceDetectionError,`	Unable to detect a face in the image
`NumDataError,`	The implementation cannot support the number of images
`TemplateFormatError,`	Template file is in an incorrect format or defective
`EnrollDirError,`	An operation on the enrollment directory failed (e.g., permission, space)
`InputLocationError,`	Cannot locate the input data – the input files or names seem incorrect
`MemoryError,`	Memory allocation failed (e.g., out of memory)
`MatchError,`	Error occurred during the 1:1 match operation
`QualityAssessmentError,`	Failure to generate a quality score on the input image
`NotImplemented,`	Function is not implemented
`VendorError`	Vendor-defined failure. Vendor errors shall return this error code and document the specific failure in the ReturnStatus.info string.
`};`	

Table 9 – ReturnStatus structure

C++ code fragment	Meaning
`struct ReturnStatus {`	
`ReturnCode code;`	Return Code
`std::string info;`	Optional information string
`// constructors`	
`};`	